EUROPEAN THEMATIC NETWORK
for DOCTORAL EDUCATION in COMPUTING
Summer School on Intelligent Systems
Nicosia, Cyprus, July 2-6, 2007

Computational Intelligence Applications in Software Engineering

Presenters:

Andreas S. Andreou
Assistant Professor

Efi Papatheocharous
PhD Candidate

Constantinos Stylianou
PhD Candidate

{aandreou, cstylianou, efi.papatheocharous}@cs.ucy.ac.cy

Department of Computer Science
University of Cyprus

COMPUTATIONAL INTELLIGENCE APPLICATIONS IN SOFTWARE ENGINEERING

Part A: Introduction to Computational Intelligence in Software Engineering

Part A: Outline

1. **Basic Software Engineering Concepts**
   - Software Development Phases
   - Complex Fundamental Problems
2. **Computational Intelligence**
   - Artificial Neural Networks
   - Genetic Algorithms
   - Fuzzy Logic
3. **Software Engineering Intelligence: Areas**

Basic Software Engineering Concepts

Software Engineering:

The process to produce quality software without defects, that is delivered on time and within budget, meeting clients' needs and can be easily maintained

Software Process:

The way we produce software, including

- Life-cycle model
- Human resources
- CASE tools

Basic Software Engineering Concepts (Cont'd)

Life-cycle model :

1. Requirements phase
2. Specification phase
3. Design phase
4. Implementation phase
5. Integration phase (parallel with 4)
6. Maintenance phase
7. Retirement

Can CI help here ?

Each phase experiences different fundamental problems

Basic Software Engineering Concepts (Cont'd)

Requirements :

- ✓ Elicitation of right needs
- ✓ Recording of all functions and constraints
- ✓ Avoidance of huge and vague documentation, etc…

Specification :

- ✓ Vagueness, ambiguity, conflicts
- ✓ Not too technical, yet quite formal
- ✓ Cost and time estimation
- ✓ SPMP (tasks, dependencies, duration, resources), etc…

Basic Software Engineering Concepts (Cont'd)

Design :

- ✓ High-level (architectural) → Logical errors, open architecture design problems, modularity, coupling, complexity of modules
- ✓ Low-level (detailed) → Complexity of data structures and variables, algorithms selection, flow of control and execution, cohesion, etc…

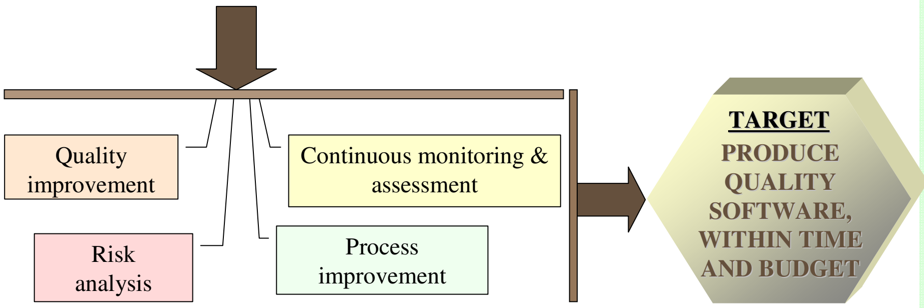
Implementation / Integration :

- ✓ Configuration management
- ✓ Programming philosophy and style
- ✓ Programming in the many/large/small
- ✓ Integration method
- ✓ Testing, etc…

Basic Software Engineering Concepts (Cont'd)

Continuous effort to improve the Software Process and Quality:

- ✓ SEI guidelines
- ✓ CMM
- ✓ SPICE
- ✓ ISO9126



Computational Intelligence

Definitions :

Conference: Computational Intelligence - Methods & Applications - CIMA2005

Defining "**Computational Intelligence**" is not straightforward. It is difficult, if not impossible, to accommodate in a formal definition disparate areas with their own established individualities such as fuzzy sets, neural networks, evolutionary computation, machine learning, Bayesian reasoning, etc.

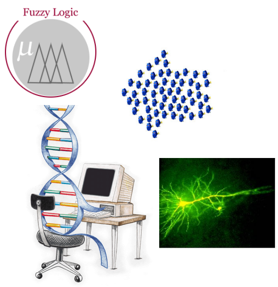
Book: “Computational Intelligence: An Introduction”, Andries P. Engelbrecht, Wiley 2002

Computational intelligence is the study of adaptive mechanisms to enable or facilitate intelligent behavior in complex and changing environments. As such, computational intelligence combines artificial neural networks, evolutionary computing, swarm intelligence and fuzzy systems.

04/07/2007 *Session 4: Computational Intelligence Applications in Software Engineering - Part A* 9

Computational Intelligence (Cont’d)

- Neural Networks
- Fuzzy Logic
- Evolutionary Algorithms
  - Genetic Algorithms
  - Genetic Programming
  - Evolutionary Programming
  - Evolutionary Strategies
  - Differential Evolution
  - Cultural Evolution, Co-evolution etc.
- Swarm Intelligence
- Case Based Reasoning
- Data Mining Techniques
- Adaptive Computing Systems
- Knowledge Based Systems
- Expert Software Systems
- Machine Learning Techniques
- Hybrid Intelligent Systems



04/07/2007 *Session 4: Computational Intelligence Applications in Software Engineering - Part A* 10

Computational Intelligence (Cont'd)

Artificial Neural Networks (ANNs) :

- ✓ Offer a powerful, distributed computing architecture that is able to learn
- ✓ Organized in layers of connected and interacting computing elements called neurons (mimic brain)
- ✓ Represent highly nonlinear, complex, multivariate relationships that are learned from experimental data
- ✓ Supervised and unsupervised learning
- ✓ Instruments for performing prediction or classification tasks

(e.g. Minsky & Pappert, 1969; Rumelhart & McClelland, 1986; Haykin, 1994)

Computational Intelligence (Cont'd)

Evolutionary computing (Genetic Algorithms - GAs) :

- ✓ Constitute a class of optimization algorithms
- ✓ A Genetic Algorithm (GA) provides a search procedure, which optimizes an objective function
- ✓ The GA maintains and evolves a population of candidate solutions through crossover and mutation operations to generate new and better, more fit individuals.
- ✓ Evolution is a stochastic process – It is based on randomness

(e.g. Fogel et al. 1966; Holland, 1975; Koza, 1990-2004)

Computational Intelligence (Cont'd)

Fuzzy Logic :

- ✓ In real world, information is often ambiguous or imprecise.
- ✓ An organized method for dealing with imprecise data is called *fuzzy logic*
- ✓ The data are considered as *fuzzy sets*. Traditional sets include or do not include an individual element; there is no other case than true or false. Fuzzy sets allow partial membership
- ✓ Fuzzy Logic is basically a multi-valued logic → allows intermediate values to be defined between conventional evaluations like *yes/no, true/false, black/white,* etc.
- ✓ Notions like *rather warm* or *pretty cold* can be formulated mathematically and processed with the computer

(e.g. Zadeh, 1965, 1973…)

Software Engineering Intelligence: Areas

Studying the needs and tackling SE problems via Computational Intelligent approaches

Research in :

1. Software Cost Estimation
2. Component-Based Software Development
3. Software Testing
4. Software Failure Modeling
5. Software Project Management
6. Software Risk Analysis & Modeling
7. Software Quality Modeling & Assessment
8. Software Reliability Modeling & Forecasting

… and more

COMPUTATIONAL INTELLIGENCE APPLICATIONS IN SOFTWARE ENGINEERING

Part B: Computational Intelligence in Software Cost Estimation

Department of Computer Science
University of Cyprus

Part B: Outline

1. **Introduction to Software Cost Estimation (SCE)**
   - Problem context and need
   - Significance and challenges
2. **Literature Review on Software Cost Estimation models**
3. **Computational Intelligence (CI) in SCE**
   - Related research carried out at the department
     - Qualitatively
       - Use of Fuzzy Cognitive Maps (FCM) in SCE
       - Cost factors are represented as concepts in the map
       - Output: Higher or lower than original estimation
     - Quantitatively
       - SCE using ANN with Inputs Selection
         - Hypothesis / Aim / Motivation
       - Description of the Datasets
       - Methodology - Design of the Experiments
       - Experimental Results
       - Conclusions
       - Future Work

Introduction to Software Cost Estimation

- Software Cost Estimation involves:
  - Prediction of the resources to be consumed in a software project.
    - Resources in terms of: costs / effort / calendar time
  - One of the most critical tasks in software engineering and project management.
- Software cost estimation takes into account:
  - Software product size
  - Functions complexity
  - Effort – measured in person months
  - Project schedules
  - Overall costs of the project

Introduction to Software Cost Estimation (Cont'd)

- Overall costs of a software project include [1]:
  - **Hardware and software costs** (including maintenance)
  - **Travel and training costs**
  - **Effort costs** (the dominant factor in most projects)
    - salaries of engineers involved in the project
    - social and insurance costs
- Effort costs must also take overheads into account:
  - costs of building, providing heating and lighting office space
  - costs of support staff such as accountants, administrators, system managers, cleaners and technicians
  - costs of networking and communications
  - costs of central facilities (e.g., library, staff restaurant, etc.)
  - costs of Social Security and employee benefits (e.g., pensions, health insurance)

[1] Sommerville, Ian. 2006. *Software Engineering:(8th Edition) (International Computer Science).* Addison-Wesley Longman Publishing Co., Inc.

Introduction to Software Cost Estimation (Cont'd)

- Estimation requirements
  - Boehm's criteria for evaluating cost models [1]:

Criteria	Description
1. Definition	Has the model clearly defined the costs it is estimating and costs it is excluding?
⇨ 2. Fidelity	Are the estimates close to the actual costs expended on the project?
3. Objectivity	Does the model avoid allocating most of the software cost variance to poorly calibrated subjective factors, such as complexity? That is, is it hard to rig the model to get the results you want?
4. Constructiveness	Can you tell a user why the model gives the estimate it does? Does it help the user understand the software job to be done?
5. Detail	Does the model easily accommodate the estimation of a software system consisting of a number of subsystems and units? Does it give (accurate) phase and activity breakdown?
6. Stability	Do small differences in inputs produce small differences in output cost estimates?
7. Scope	Does the model cover the class of software projects whose costs you need to estimate?
8. Ease of use	Are the model inputs and options easy to understand and specify?
⇨ 9. Prospectiveness	Does the model avoid the use of information which will not be known until the project is complete?
10. Parsimony	Does the model avoid the use of highly redundant factors or factors which make no appreciable contribution to the result?

[1] Boehm, BW. 1981. *Software Engineering Economics*. Prentice Hall PTR Upper Saddle River, NJ, USA.

    - Accuracy
    - Early in the development life-cycle

Importance of Software Cost Estimation

- Why is it important to estimate software costs accurately and early in the development cycle?
  - For the successful delivery of a software product on time, within budget and with the anticipated functionality
  - For reducing risks, uncertainty, supporting better decision making
  - Better project management

- Some inherent problems of software production:
  - Complexity
  - Conformability
  - Changeability
  - Invisibility
  - → *"Software is invisible and unvisualizable."* Frederick P. Brooks
  - → All contribute to making Software Cost Estimation <u>hard</u>.

Boehm's Cone of Uncertainty (ref.[1])

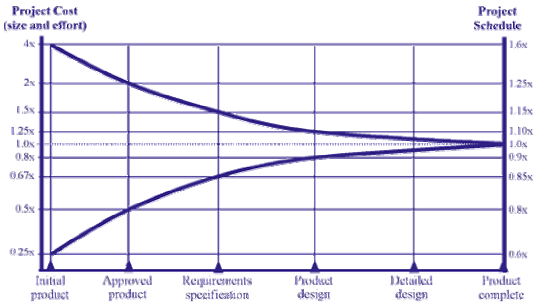


Figure 1: Project Cost Estimates during a sequential development process [2].

- Displays the uncertainty in the estimates at each stage of the project
- Early project estimates will inevitably be highly inaccurate
- Total estimate range is 16x (best-case scenario)
- Could easily become the 'cloud of uncertainty' [3]

[1] Boehm, BW. 1981. *Software Engineering Economics.* Prentice Hall PTR Upper Saddle River, NJ, USA.
[2] McConnell, Steve, "10 Deadly Sins of Software Estimation", Available from http://www.construx.com, Accessed 2007.
[3] McConnell, Steve. 2006. *Software Estimation: Demystifying the Black Art (Best Practices).* Microsoft Press.

Challenges in Early Software Cost Estimation

- Why is the estimate hard?
  - High complexity and uniqueness of the software engineering process
  - Dynamic parameters affecting productivity and effort
  - Never two software systems or projects are identical
  - May need to run on unfamiliar computers, or use new technologies
  - Undergo new processes
  - Different people may be involved in the development process
    - Having different skills, culture, experiences, knowledge
  - Limited knowledge and experience concerning the relationships between the factors affecting software productivity and effort
  - Lack of trained estimators with the necessary expertise and knowledge to support the estimation process
  - Low number of active researchers with long-term interest on software cost estimation compared to the researched topics and approaches

Some facts about the software industry

- The Software Productivity Research LLC surveyed 250 large software projects (during the period 1995 – 2004)
  - Software projects tend to have:
    - A very high frequency of schedule overruns, cost overruns, quality problems or even cancellations [3]
    - Poor project planning, poor cost estimating, poor measurements, poor milestone tracking, poor change control, and poor quality control [2]
- The Standish Group [5] surveyed over 40,000 projects in 10 years to reach the following findings [4]:
    - 23% of all software projects are cancelled before completion
    - Only 28% of the completed projects are delivered on time, within budget and with all originally specified features
    - The average software project cost overruns budget by 45%

[1] Jones, Capers; "Software Estimating Methods for Large Projects"; Crosstalk, April 2005.

[2] Jones, Capers. 2004. Software Project Management Practices: Failure Versus Success©. Crosstalk 17, no. 19: 5-9.

[3] Jones, Capers. 2005. "How software estimation tools work". Software Productivity Research 1996 - 2005 by Capers Jones, Chairman, SPR, Inc.

[4] Laird, Linda M., and M. Carol Brennan. 2006. *Software Measurement and Estimation: A Practical Approach (Quantitative Software Engineering Series).* Wiley-IEEE Computer Society Press.

[5] The Standish Group, CHAOS Chronicles, Standish Group Internal Report, 1995, Available at <http://www.standishgroup.com/>.

Some challenges: History of IT projects

- Software projects often fail [1].

YEAR	COMPANY	OUTCOME (COSTS IN US $)
2005	Hudson Bay Co. [Canada]	Problems with inventory system contribute to $33.3 million* loss.
2004–05	UK Inland Revenue	Software errors contribute to $3.45 billion* tax-credit overpayment.
2004	Avis Europe PLC [UK]	Enterprise resource planning (ERP) system canceled after $54.5 million† is spent.
2004	Ford Motor Co.	Purchasing system abandoned after deployment costing approximately $400 million.
2004	J Sainsbury PLC [UK]	Supply-chain management system abandoned after deployment costing $527 million.†
2004	Hewlett-Packard Co.	Problems with ERP system contribute to $160 million loss.
2003–04	AT&T Wireless	Customer relations management (CRM) upgrade problems lead to revenue loss of $100 million.
2002	McDonald's Corp.	The Innovate information-purchasing system canceled after $170 million is spent.
2002	Sydney Water Corp. [Australia]	Billing system canceled after $33.2 million† is spent.
2002	CIGNA Corp.	Problems with CRM system contribute to $445 million loss.
2001	Nike Inc.	Problems with supply-chain management system contribute to $100 million loss.
2001	Kmart Corp.	Supply-chain management system canceled after $130 million is spent.
2000	Washington, D.C.	City payroll system abandoned after deployment costing $25 million.

SOURCES: *BUSINESS WEEK, CEO MAGAZINE, COMPUTERWORLD, INFOWEEK, FORTUNE, THE NEW YORK TIMES, TIME,* AND *THE WALL STREET JOURNAL*

- Why?
  - One of the most common reasons for project failure is the inaccurate estimate of the needed resources.
  - → Project managers stress the importance of having supportive methods to estimate software costs

[1] Charette, R.N. 2005. Why software fails [software failure]. *Spectrum, IEEE* 42, no. 9: 42- 49.

Background on Cost Estimation Models

- Types of Cost estimation models:
  - Cost-oriented - provide direct estimates of effort
  - Constraint-oriented – express relationship between parameters and effort over time

- Estimation Techniques [1]

Technique	Description
Algorithmic cost modelling	A model is developed using historical cost information that relates some software metric (usually its size) to the project cost. An estimate is made of that metric and the model predicts the effort required.
Expert judgement	Several experts on the proposed software development techniques and the application domain are consulted. They each estimate the project cost. These estimates are compared and discussed. The estimation process iterates until an agreed estimate is reached.
Estimation by analogy	This technique is applicable when other projects in the same application domain have been completed. The cost of a new project is estimated by analogy with these completed projects. Myers (Myers, 1989) gives a very clear description of this approach.
Parkinson's Law	Parkinson's Law states that work expands to fill the time available. The cost is determined by available resources rather than by objective assessment. If the software has to be delivered in 12 months and 5 people are available, the effort required is estimated to be 60 person-months.
Pricing to win	The software cost is estimated to be whatever the customer has available to spend on the project. The estimated effort depends on the customer's budget and not on the software functionality.

[1] Sommerville, Ian. 2006. *Software Engineering:(8th Edition) (International Computer Science)*. Addison-Wesley Longman Publishing Co., Inc.

Background on Cost Estimation Models (Cont'd)

- Algorithmic Cost Modeling
  - Use a mathematical formula to predict project costs based on estimates of the project size, the number of software engineers, and other process and product factors
  - It can be built by analysing the costs and attributes of completed projects and finding the closest fit formula to actual experience
    - Effort = $A \times Size^{B} \times M$
    - A is an organisation-dependent constant (local practices, type of software to be developed etc.), B reflects the disproportionate effort for large projects and M is a multiplier reflecting product, process and people attributes
    - The most commonly used product attribute for cost estimation is the code Size and it is usually measured in Lines Of Code, Function Points, Object Points
  - Limitations:
    - Difficult and subjective to estimate Size at an early project stage
    - Subjective measures of A, B and M

Background on Cost Estimation Models (Cont'd)

- The COCOMO Model
  - Empirical Algorithmic Model
  - Derived by collecting data from a large number of software projects
  - Discover a formula that link the size of the system and product, project and team factors to the effort to develop the system
  - Characteristics:
    - It is well documented, available in the public domain and supported by public domain and commercial tools
    - It has been widely used and evaluated in a range of organisations
    - Redefined over the years from the initial version COCOMO I [1] to a recent version COCOMO II [2]

[1] Boehm, B.W., 1981. *Software Engineering Economics*. Prentice Hall.

[2] Boehm, B. 2000. Safe and simple software cost analysis. Software, IEEE 17, no. 5: 14-17.

Background on Cost Estimation Models (Cont'd)

- COCOMO I

Project complexity	Formula	Description
Simple	$PM = 2.4\ (KDSI)^{1.05} \times M$	Well-understood applications developed by small teams
Moderate	$PM = 3.0\ (KDSI)^{1.12} \times M$	More complex projects where team members may have limited experience of related systems
Embedded	$PM = 3.6\ (KDSI)^{1.20} \times M$	Complex projects where the software is part of a strongly coupled complex of hardware, software, regulations and operational procedures

KDSI : Thousands of Delivered Source Instructions

M : Multipliers are created and adjusted according to project cost drivers

- COCOMO II

1. Early Prototyping Level
   - Estimates based on Object Points and a simple formula is used for effort estimation (draft requirements+prototyping)
2. Early Design Level
   - Estimates based on FP that are then translated to LOC (full reqs+specs, perhaps some initial design)
3. Post-architecture Level
   - Estimates based on LOC

Background on Estimation models

- COCOMO (continued)
  - Project cost drivers

Attribute	Type	Description
RELY	Product	Required system reliability
CPLX	Product	Complexity of system modules
DOCU	Product	Extent of documentation required
DATA	Product	Size of database used
RUSE	Product	Required percentage of reusable components
TIME	Computer	Execution time constraint
PVOL	Computer	Volatility of development platform
STOR	Computer	Memory constraints
ACAP	Personnel	Capability of project analysts
PCON	Personnel	Personnel continuity
PCAP	Personnel	Programmer capability
PEXP	Personnel	Programmer experience in project domain
AEXP	Personnel	Analyst experience in project domain
LTEX	Personnel	Language and tool experience
TOOL	Project	Use of software tools
SCED	Project	Development schedule compression
SITE	Project	Extent of multisite working and quality of inter-site communications

Return

Background on Cost Estimation Models (Cont'd)

- COCOMO II - Formulas
  1. Early Prototyping Level
     - PM = (NOP × (1 - %reuse/100)) / PROD
     - PM is the effort in person-months, NOP is the number of object points and PROD is the productivity
  2. Early Design Level
     - PM = A × Size^B × M + PM$_m$ where M = PERS×RCPX×RUSE×PDIF×PREX×FCIL×SCED
     - PM$_m$ = (ASLOC × (AT/100)) / ATPROD
     - PM$_m$ is a factor used when code is generated automatically, with ASLOC being the number of automatically generated LOC, AT the percentage of total system code automatically generated, and ATPROD the productivity level for automatic code
  3. Post-architecture Level
     - ESLOC = ASLOC × (AA + SU +0.4DM + 0.3CM +0.3IM)/100
     - ESLOC is equivalent number of lines of new code. ASLOC is the number of lines of reusable code which must be modified, DM is the percentage of design modified, CM is the percentage of the code that is modified , IM is the percentage of the original integration effort required for integrating the reused software. SU is a factor based on the cost of software understanding, AA is a factor which reflects the initial assessment costs of deciding if software may be reused

Computational Intelligence in SCE

- There is need for intelligent methods to support software cost estimations
- Computational Intelligence seems to provide optimal solutions to complex problems, and combine elements of learning, adaptation and evolution

Study (Year)	Comparing Approach	Dataset	Results
Dolado (1998)	Regression Neural Networks Genetic Algorithms*	Belady COCOMO Albrecht Kemerer Acad. Env. Matson	MMRE=27% MMRE=22%
Shukla (2000)	CART Neural Networks (backprop) Neural Networks (quickprop) Hybrid(NN&GA)*	COCOMO& Kemerer	R^2 =0.9979
Mair (2000)	Neural Networks* CBR RI	Deshamais	MMRE=21%
Burgess (2001)	Neural Networks Genetic Programming*	Deshamais	MMRE=37%
Idri (2002)	Neural Networks Fuzzy Logic	COCOMO	MMRE=203% MMRE=84% Increase#projects Incr. Interpretability
Xu (2004)	Fuzzy cross Fuzzy fit	COCOMO	MMRE=13%
Martin (2005)	Fuzzy logic		MMRE=0.1057
Huang (2006)	Hybrid Soft Computing (Neural Networks, Fuzzy, Algorithmic-Expert)		Improvement=13%

Computational Intelligence in SCE (Cont'd)

- Research Topics and Estimation Approaches [1]

Research topic	-1989	1990-1999	2000-2004	Total
Em	30 (73%)	96 (59%)	58 (58%)	184 (61%)
Pf	8 (20%)	7 (4%)	3 (3%)	18 (6%)
Cm	3 (7%)	13 (8%)	4 (4%)	20 (7%)
Sm	5 (12%)	39 (24%)	16 (16%)	60 (20%)
Oi	9 (22%)	25 (15%)	14 (14%)	48 (16%)
Un	2 (5%)	10 (6%)	13 (13%)	25 (8%)
Ep	2 (5%)	8 (5%)	6 (6%)	16 (5%)
Ds	0 (0%)	1 (1%)	2 (2%)	3 (1%)
Ot	0 (0%)	3 (2%)	1 (1%)	4 (1%)

The abbreviations used are: Estimation method = Em, Production function = Pf, Calibration of models = Cm, Size measures = Sm, Organizational issues = Oi, Uncertainty assessments = Un, Measures of estimation performance = Ep, Data set properties = Ds, Other = Ot.

Estimation approach	-1989	1990-1999	2000-2004	Total
Rg	21 (51%)	76 (47%)	51 (51%)	148 (49%)
An	1 (2%)	15 (9%)	15 (15%)	31 (10%)
Ej	3 (7%)	22 (13%)	21 (21%)	46 (15%)
Wb	3 (7%)	5 (3%)	4 (4%)	12 (4%)
Fp	7 (17%)	47 (29%)	14 (14%)	68 (22%)
Ct	0 (0%)	5 (3%)	9 (9%)	14 (5%)
Si	2 (5%)	4 (2%)	4 (4%)	10 (3%)
Nn	0 (0%)	11 (7%)	11 (11%)	22 (7%)
Th	20 (49%)	14 (9%)	5 (5%)	39 (13%)
By	0 (0%)	1 (1%)	6 (6%)	7 (2%)
Cb	0 (0%)	3 (2%)	2 (2%)	5 (2%)
Ot	2 (5%)	7 (4%)	16 (16%)	25 (8%)

The abbreviations used are: Rg = Regression, An = Analogy, Ej = Expert judgment, Wb = Work break-down, Fp = Function Point, Ct = Classification and regression trees, Si = Simulation, Nn = Neural network, Th = Theory, By = Bayesian, Cb = Combination of estimates, Ot = Other

- There is no silver bullet
- Each approach contains limitations

→ *"Software Estimation has been identified as one of the three great challenges for half-century-old in computer science."*

[1] Jorgensen, M., and M. Shepperd. 2007. A Systematic Review of Software Development Cost Estimation Studies. *Software Engineering, IEEE Transactions on* 33, no. 1: 33-53.

SCE using ANN with Inputs Selection (ref. [1])

- Recent work: Identify and select best suited project attributes
- Hypothesis:
  - Identify critical project characteristics
  - Evaluate their impact on the evolution of software cost
  - → Could provide more accurate estimations
- Aim: Accurately predict software development cost
  - Computational Intelligent (CI) methods
  - Input Sensitivity Analysis (ISA)
  - → Find the optimal set of input parameters in order to:
    - describe better the cost of a software project
    - in earlier phases of the software development life-cycle (SDLC)
- Motivation:
  - If a satisfactory and reliable model is devised it can constitute the basis for:
    - contract negotiations, project charging, classification of tasks, allocation of human resources, monitoring task progress, etc.

[1] E. Papatheocharous, A. Andreou, "Software Cost Estimation Using Artificial Neural Networks with Inputs Selection", *Proc. of the 9th International Conf. on Enterprise Information Systems, pp.398-407, ICEIS* Madeira, 2007.

Description of Datasets

- Desharnais (1988)
  - ~80 systems developed by a Canadian software development house
- ISBSG (Release 9)
  - International Software Benchmarking Standards Group
    - Broad cross range data (multi-organisational, multi-application domain, multi-environmental)

Id	Table 1. Desharnais Attributes
1	Project Name
2	Team's Experience (Years)
3	Project Manager's Experience
4	Length Of Development
5	Development Effort (Hours)
6	Number Of Transactions
7	Number Of Entities
8	Unadjusted Function Points
9	Scope
10	Adjusted Function Points
11	Language

Id	Table 2. ISBSG Attributes
1	Project Name
2	Functional Size
3	Adjusted Function Points
4	Unadjusted Function Points
5	Project Elapsed Time
6	Project Inactive Time
7	Count Approach
8	Normalised Work Effort
9	Productivity Delivery Rate
…	
79	Lines of Code

Artificial Neural Networks (ANN)

- A simple neuron definition:
  - The basic unit of an ANN, simulating a biological neuron.
  - Inspiration originates from the desire to model the way the human brain works and create sophisticated artificial systems that are capable of intelligent computations, similar to the computations of the biological neurons in the brain structures.

- Structure of a simple neuron
  - One or more Inputs
  - Weights
  - Activation function
    - Sigmoid $y = \frac{1}{1+e^{-x}}$
    - Gaussian
    - Etc.

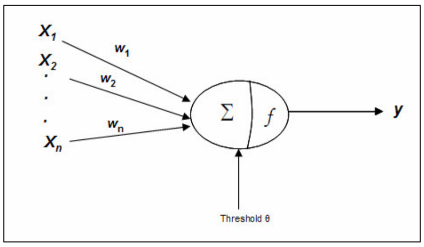
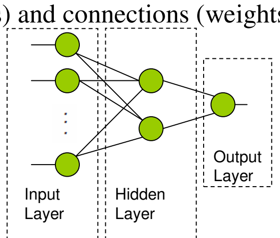


Figure 2: Structure of a simple neuron

Artificial Neural Networks (ANN) (Cont'd)

- ANN definition:
  - An ANN can be viewed as a directed graph.
  - It is composed of a number of basic computational elements (called neurons or nodes) and connections (weights) between them, forming layers.



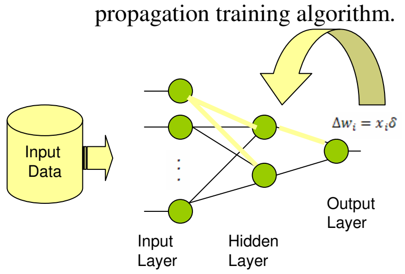
  - **Supervised** Vs Unsupervised learning
    - It requires a desired output in order to learn
    - It has the ability to:
      - Represent complex relationships
      - Identify patterns
      - Learn and Generalise the acquired knowledge

Artificial Neural Networks (ANN) (Cont'd)

- A single layer Perceptron Model of McCulloch-Pitts [1]:
  - Consists of:
    - A set of inputs weights
    - A threshold
    - A hard limiter
  - The hidden layers provide connectivity between the inputs and outputs.
- The network may also have feedback, which will take result variables and use them as input to prior processing nodes
- Feed-forward Multi-Layer Perceptron (MLP)
  - Consists of multiple layers of computational units
  - Interconnected in a feed-forward way
  - Each neuron in one layer has directed connections to the neurons of the subsequent layer

Artificial Neural Networks (ANN) (Cont'd)

- Software models assessing cost or effort use:
  - Feed-forward MLP form of ANN, supervised learning methods and back-propagation training algorithm.




```
1. Initialize the weights in the network (often randomly)
2. repeat
       * foreach example e in the training set do
            1. O = neural-net-output(network, e) ; forward pass
            2. T = teacher output for e
            3. Calculate error (T - O) at the output units
            4. Compute delta_wi for all weights from hidden layer to output layer ; backward pass
            5. Compute delta_wi for all weights from input layer to hidden layer ; backward pass continued
            6. Update the weights in the network
       * end
3. until all examples classified correctly or stopping criterion satisfied
4. return(network)
```

Methodology - Design of the Experiments

- Step 1: Data Pre-processing
  - Null values
  - Incomplete data
  - Normalization [-1, 1]
- Step 2: Iterative process
  - Data Sampling:
    - 70% training / 20% validation / 10% testing
  - Use Feed-forward Multi-Layer Perceptron ANN with varying hidden neurons
  - Select 20% of the best ANNs based on performance metrics
  - Apply Inputs Sensitivity Analysis (ISA)
    - Sum input weights
    - Identify important inputs
      - Strict (S) Threshold: $w_{th}^{S} = \frac{\max(w_1 - w_n) - \min(w_1 - w_n)}{2}$
      - Less Strict (LS) Threshold: $w_{th}^{LS} = \max(w_1 - w_n) * 0{,}25$
    - Estimate inputs acceptance percentage rate $rate\_total_i = \frac{num\_of\_ANNs\_W_{th,i}^{S\ or\ LS}}{total\_number\_ANNs} \%$
- Step 3: Derive Final Parameters (FP) set

Evaluation of ANN's Performance

- Relative Mean Absolute Error (RMAE) $RMAE(n) = \frac{\frac{1}{n}\sum_{i=1}^{n}\left|x_{act}(i) - x_{pred}(i)\right|}{x_{act}(i)}$
- Correlation Coefficient (CC) $CC = \frac{\sum_{i=1}^{n}\left[\left(x_{act}(i) - \bar{x}_{act,n}\right) - \left(x_{pred}(i) - \bar{x}_{pred,n}\right)\right]}{\sqrt{\left[\sum_{i=1}^{n}\left(x_{act}(i) - \bar{x}_{act,n}\right)^2\right]\left[\sum_{i=1}^{n}\left(x_{pred}(i) - \bar{x}_{pred,n}\right)^2\right]}}$
- Normalized Root Mean Squared Error (NRMSE) $NRMSE(n) = \frac{RMSE(n)}{\sigma_\Delta} = \frac{RMSE(n)}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[x_{act}(i) - \bar{x}_n\right]^2}}$
- Where, $RMSE(n) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[x_{pred}(i) - x_{act}(i)\right]^2}$
- Pred(l)
  - How many data predictions k out of n (total number of data points predicted) performed well, i.e., their RE metric given in equation is lower than level *l:* $pred(l) = \frac{k}{n}$
  - Relative Error (RE): $RE(n) = \frac{\left|x_{act}(i) - x_{pred}(i)\right|}{x_{act}(i)}$

Experimental Results Desharnais

- Desharnais Indicative Results (Performance Errors vs Average Weights vs Threshold Acceptance)

ANN Experimental Results / Training And Testing Errors (Indicative runs – out of 10 iterations)

ANN.	TRAINING						TESTING					
	NRMSE	CC	MSE	RMAE	MAE	pred(l)	NRMSE	CC	MSE	RMAE	MAE	pred(l)
9-20-1	0.2328	0.9720	0.0058	0.1012	0.0592	0.9811	0.3289	0.9552	0.0026	0.0470	0.0366	1
9-16-1	0.2489	0.9682	0.0066	0.1033	0.0597	1	0.3666	0.9325	0.0033	0.0610	0.0440	1
9-10-1	0.3055	0.9514	0.0100	0.1235	0.0729	0.9811	0.3777	0.9215	0.0035	0.0634	0.0504	1
9-19-1	0.1797	0.9834	0.0034	0.0709	0.0453	1	0.4661	0.8997	0.0053	0.0735	0.0600	1

ANN Experimental Results / Average Weights for each Input

ANN	Team Exp.	Manager Exp.	Length	Transactions	Entities	Points adj.	Envergure	Points non adj.	Language
9-20-1	0.0658	0.0910	0.0238	0.0606	**0.2068**	0.0790	0.0577	**0.1495**	0.0658
9-16-1	0.1232	0.0109	0.0354	0.2124	0.0633	**0.4583**	0.0290	0.0300	0.1232
9-10-1	0.1096	0.1718	0.0699	0.0845	**0.2855**	**0.3958**	0.0698	**0.1939**	0.1096
9-19-1	0.0855	0.0176	0.1947	0.0951	**0.2519**	0.1112	0.1179	0.0312	0.0855

ANN Experimental Results Input Sensitivity Analysis / Strict (S) and Less Strict (LS) Approach

ANN	Team Exp.		Manager Exp.		Length		Transactions		Entities		Points adj.		Envergure		Points non adj.		Language	
	S	LS	S	LS	S	LS	S	LS	S	LS	S	LS	S	LS	S	LS	S	LS
9-20-1		✓		✓				✓	✓	✓		✓		✓	✓	✓		
9-16-1		✓						✓			✓	✓						✓
9-10-1		✓	✓					✓	✓		✓	✓			✓			✓
9-19-1		✓			✓	✓		✓	✓	✓		✓		✓				

Desharnais Dataset Final Parameter Set Selected

FP(Desharnais)=

Strict	Less Strict	'Early'
Points adjusted	Points adjusted	Points adjusted
Points non adjusted	Points non adjusted	Points non adjusted
	Team Experience	Team Experience
	Transactions	Manager Experience
	Entities	

Experimental Results ISBSG

- ISBSG Indicative Results (Performance Errors vs Average Weights vs Threshold Acceptance)

ANN Experimental Results / Training And Testing Errors (Indicative runs – out of 10 iterations)

ANN	TRAINING						TESTING					
	NRMSE	CC	MSE	RMAE	MAE	pred(l)	NRMSE	CC	MSE	RMAE	MAE	pred(l)
18-22-1	0.3857	0.9237	0.0020	0.0349	0.0294	1	0.3761	0.9381	0.0044	0.0642	0.0474	1
18-23-1	0.4307	0.9026	0.0026	0.0355	0.029	0.9980	0.3764	0.9509	0.0048	0.0541	0.0430	0.9932
18-35-1	0.4934	0.8727	0.0034	0.0434	0.0373	0.9980	0.3587	0.9482	0.0044	0.0532	0.0432	0.9932
18-19-1	0.3294	0.9452	0.0015	0.0300	0.0256	1	0.2616	0.9675	0.0025	0.0409	0.0328	1

ANN Experimental Results / Average Weights for each Input

ANN	Reported pdr (afp)	Project pdr (ufp)	Normalized pdr (afp)	Normalized pdr (ufp)	Input count	Enquiry count	Interface count	Added count	Deleted count
18-22-1	0.10	0.02	0.086	0.01	0.05	0.14	0.016	0.11	0.08
18-23-1	0.09	0.18	0.153	0.13	0.05	0.07	0.131	0.06	0.01
18-35-1	0.04	0.03	0.013	0.01	0.08	0.06	0.088	0.11	0.01
18-19-1	0.00	0.15	0.185	0.18	0.11	0.18	0.011	0.15	0.22

ANN Experimental Results Input Sensitivity Analysis / Strict (S) and Less Strict (LS) Approach

ANN	Reported pdr (afp)		Project pdr (ufp)		Normalized pdr (afp)		Normalized pdr (ufp)		Input count		Enquiry count		Interface count		Added count		Deleted count	
	S	LS	S	LS	S	LS	S	LS	S	LS	S	LS	S	LS	S	LS	S	LS
18-22-1	✓	✓			✓	✓				✓	✓	✓			✓	✓	✓	✓
18-23-1	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	✓		✓		
18-35-1		✓							✓	✓	✓	✓	✓	✓	✓	✓		
18-19-1			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓

ISBSG Dataset Final Parameter Set Selected

FP(ISBSG)=

Strict	Less Strict	'Early'
Normalised PDR (afp)	Normalised PDR (afp)	Functional size
File count	File count	Adjusted Function Points
Added count	Added count	Normalised PDR (afp)
	Enquiry count	
	Changed count	

Final Runs – Validation with the FP set

Desharnais dataset ANN Final Runs					
TESTING					
NRMSE	CC	MSE	RMAE	MAE	pred(l)
Strict (S) Approach					
0.6916	0.6999	0.0169	0.1956	0.0907	1
0.7172	0.6762	0.0181	0.2089	0.1027	1
0.8602	0.5801	0.0261	0.2431	0.1125	1
0.7202	0.6759	0.0183	0.2026	0.0905	1
Less Strict (LS) Approach					
0.6550	0.8330	0.0151	0.1844	0.0806	1
0.6967	0.7719	0.0171	0.1875	0.0767	1
0.6104	0.8410	0.0131	0.1718	0.0826	1
0.6414	0.8040	0.0145	0.1780	0.0781	1
"Ideal" (Early Phase) Approach					
0.8772	0.4852	0.0272	0.2463	0.1108	1
0.7052	0.7405	0.0175	0.2044	0.0932	1
0.7419	0.6769	0.0194	0.2095	0.0945	1
0.8390	0.5076	0.0248	0.2370	0.1349	1

ISBSG dataset ANN Final Runs					
TESTING					
NRMSE	CC	MSE	RMAE	MAE	pred(l)
Strict (S) Approach					
0.5246	0.8523	0.0077	0.0374	0.0350	0.9932
0.2195	0.9766	0.0013	0.0260	0.0242	1
0.4290	0.9086	0.0052	0.0418	0.0390	0.9932
0.3564	0.9386	0.0035	0.0390	0.0360	1
Less Strict (LS) Approach					
0.3031	0.9565	0.0026	0.0384	0.0355	1
0.5229	0.8617	0.0077	0.0445	0.0418	0.9932
0.5984	0.8055	0.0101	0.0389	0.0360	0.9932
0.6938	0.7178	0.0136	0.0495	0.0457	0.9932
"Ideal" (Early Phase) Approach					
0.2968	0.9559	0.0024	0.0392	0.0363	1
0.2355	0.9731	0.0015	0.0258	0.0239	1
0.1962	0.9805	0.0010	0.0232	0.0218	1
0.2805	0.9602	0.0022	0.0276	0.0257	1

- Reducing the number of attributes we achieved low performance error values
- Results are more than promising – there is slight rise in the performance metrics
- Fairly similar results during testing
- Relatively similar predictive power of the initial and the reduced FP set

→ Achieved to reduce the necessary number of attributes in the estimates
→ Using attributes that can be measured early can produce accurate cost estimations

Conclusions

- Consistent results in the parameters selected by both S and LS
  - Highly important inputs (Desharnais):
    - Points Adjusted
  - Highly important inputs (ISBSG):
    - Normalised PDR (afp)
  - → Software Size and Product Delivery Rate are the highly significant cost drivers
- Isolated significant parameters for new experiments
  - Results indicated highly accurate effort estimates
    - Minimised the number of parameters used
    - An average of 3 to 5 specific parameters is sufficient
- Applied 'early' estimate
  - Similarly successful estimates as before
- Overall benefit of the methodology:
  - Higher efficiency, Lower complexity, High accuracy
  - Early estimation
  - Identify parameters that decisively influence the evolution of software cost

Future Work

- Further investigation of the approach proposed
  - Eliminate limitations of the approach
  - Apply the approach on other datasets
  - Examine the consistency of the FP set
- Assess the importance of cost factors with other methods
  - Investigate other Computational Intelligence Techniques (e.g., Genetic Algorithms, Fuzzy Logic)
- Incorporate the model in a real-life software cost estimation environment
  - assess the degree to which a set of inputs measured under the same software development conditions, team and project characteristics may derive consistent dependencies to software costs

COMPUTATIONAL INTELLIGENCE APPLICATIONS IN SOFTWARE ENGINEERING

Part C: Computational Intelligence in Component-Based Software Engineering

Department of Computer Science
University of Cyprus

Part C: Outline

1. Introduction
   - Problem statement
   - Goals
   - Previous attempts
2. Clustering algorithms
   - Notations
   - Fuzzy *k*-modes clustering
   - Entropy-based clustering
3. Methodology
   - Evolution
   - Description
   - Demonstration
   - Evaluation
4. Concluding remarks
   - Synopsis
   - Pros and cons
   - Future work

Introduction

- Problem statement
- Goals
- Previous attempts

Problem Statement

- Component-based software development process:
  - constructing large and often complex systems from smaller, autonomous and reusable software units called software components.
- 4 Steps:
  - Component qualification (suitability testing)
    - Discovery and evaluation
  - Component adaptation
    - Configuration
  - Assembling components into systems
    - Integration
  - System evolution
    - Maintenance

Goals

- Aim
  - Improve the component-based development process
- How?
  - Shorten the process' development time
- Where?
  - Component qualification → discovery
- Requirements
  - For searching: efficiency
  - For retrieving: adequateness
- Method?
  - Cluster components in the repository into subsets
  - Find the nearest subset to the user's search preference
  - Retrieve most suitable from in there

Previous Attempts

- Informal
  - Facets (Pietro-Diaz, 1987, 1991)
  - Free-text analysis for automatically extracting keywords (Girardi, 1995)
  - Semantic networks (Sugumaran, 2003; Yao, 2004)
- Formal
  - Specification-based (Chu, 2000; Nakkrasae, 2003 ; Nakkrasae, 2004)
  - Modelling artefacts (Chang, 2005)
- Self-organising maps (Wang, 2004)
- Genetic algorithms (Andreou, 2004)

Clustering Algorithms

- Notations
- Clustering approach used by (Tsekouras, 2004) employs:
  - Entropy-based clustering (Yao, 2003)
  - Fuzzy *k*-modes clustering (Huang, 1998)

Notations

- A dataset $X = \{X_1, X_2, ..., X_n\}$ consists of n objects
- Each of these objects can be defined by a set of attributes, $A_1, A_2, ..., A_m$ and attribute A_j can take any value from its domain, $\mathrm{DOM}(A_j)$, for $1 \le j \le m$
- The dataset can therefore be logically viewed as a conjunction of attribute-value pairs $x_{i,j} \in \mathrm{DOM}(A_j)$ $[A_{i,1} = x_{i,1}] \wedge [A_{i,2} = x_{i,2}] \wedge ... \wedge [A_{i,m} = x_{i,m}]$, where
- A cluster is a representative of a subset of data and is denoted by $Z_l = \{z_{l,1}, z_{l,2}, ..., z_{l,m}\}$

Entropy-based Clustering

- Basic idea
  - Groups similar data objects together into clusters based on data objects' entropy values using a similarity measure
- Passes through the dataset only once
- Requires a threshold of similarity parameter
- Can be used to compute the number of clusters in a dataset as well as to find the locations of cluster centres

Entropy-based Clustering (Cont'd)

- Algorithm:
  - For each data object, calculate its entropy value based on

$$E_i = -\sum_{\substack{j=1 \\ i\neq k}}^{n}\left[S_{ij}\log_2(S_{ij}) - (1-S_{ij})\log_2(1-S_{ij})\right] \quad S_{kl} = e^{-\alpha D_{ij}}$$

  - The data object achieving the lowest entropy value is selected the first cluster centre
  - Data objects with high similarity to the recently selected cluster centre (i.e., data objects with a similarity value higher than the threshold) are removed from the dataset.
  - Once these data objects are removed from the dataset, the number of clusters is increased and the data object with the next least entropy value is selected and the procedure repeats until there are no objects left in the dataset.

Entropy-based Clustering (Cont'd)

Algorithm: **Entropy-based clustering**

1. Select threshold of similarity, β and set the initial number of clusters $c = 0$.
2. Determine the total entropy values H for each data object in X.
3. Set $c = c + 1$.
4. Select the data object $X_{\min}$ with the least entropy $H_{\min}$ and set $Z_c = X_{\min}$ as the c^{th} cluster centre.
5. Remove $X_{\min}$ and all data objects having similarity with $X_{\min}$ greater than β from X.
6. If X is empty then stop; otherwise go to step 3.

Entropy-based Clustering (Cont'd)

X_1

$X_2 \rightarrow D_2 \rightarrow S_2 \rightarrow E_2$

$X_3 \rightarrow D_3 \rightarrow S_3 \rightarrow E_3$

$\vdots \quad \vdots \quad \vdots \quad \vdots$

$X_n \rightarrow D_n \rightarrow S_n \rightarrow E_n$

$\downarrow$

E_{total}

Entropy-based Clustering (Cont'd)

X	1	2	3	4	5	6	7	8	9	10	11	12
E_{total}	44.59	42.63	43.32	43.69	42.74	44.01	44.34	43.19	43.28	42.98	43.12	44.13

- Similarity of objects using a threshold $\beta = 0.50$

C1

C2

C3

C4

- $k = 4$ with cluster centres = {2, 5, 11, 6}

Entropy-based Clustering (Cont'd)

X	1	2	3	4	5	6	7	8	9	10	11	12
1	0.00	0.36	0.22	0.38	0.52	0.46	0.68	0.46	0.64	0.43	0.10	0.49
2	0.36	0.00	0.31	0.72	0.38	0.36	0.38	0.72	0.36	0.68	0.33	0.43
3	0.22	0.31	0.00	0.38	0.46	0.46	0.46	0.36	0.41	0.36	0.88	0.46
4	0.38	0.72	0.38	0.00	0.46	0.49	0.36	0.72	0.41	0.64	0.38	0.46
5	0.52	0.38	0.46	0.46	0.00	0.46	0.56	0.38	0.77	0.36	0.26	0.41
6	0.46	0.36	0.46	0.49	0.46	0.00	0.43	0.43	0.20	0.43	0.46	0.68
7	0.68	0.38	0.46	0.36	0.56	0.43	0.00	0.41	0.64	0.36	0.32	0.49
8	0.46	0.72	0.36	0.72	0.38	0.43	0.41	0.00	0.38	0.82	0.38	0.46
9	0.64	0.36	0.41	0.41	0.77	0.20	0.64	0.38	0.00	0.38	0.26	0.41
10	0.43	0.68	0.36	0.64	0.36	0.43	0.36	0.82	0.38	0.00	0.38	0.43
11	0.10	0.33	0.88	0.38	0.26	0.46	0.32	0.38	0.26	0.38	0.00	0.33
12	0.49	0.43	0.46	0.46	0.41	0.68	0.49	0.46	0.41	0.43	0.33	0.00

Fuzzy *k*-Modes Clustering

- Basic Idea
  - Create a finite number (*k*) of partitions of the objects within a dataset so that it maximises the similarity of objects within a partition but at the same time keeps the similarity of objects between partitions to a minimum.
- Requires:
  - Finite number *k* (≤ n)
  - Dissimilarity measure
  - Updating function

Fuzzy *k*-Modes Clustering (Cont'd)

- *k*-Modes Algorithm:
  - Select *k* initial cluster centres randomly
  - Assign each data object to the cluster it is most similar to using a dissimilarity measure
  - After all objects have been assigned, update the new cluster centres based on the frequency of categories of attributes (modes of attributes)
  - The algorithm repeats until there is no change in the (re)assignment of objects or the location of the cluster centres

Fuzzy *k*-Modes Clustering (Cont'd)

- Fuzzy *k*-Modes Algorithm:
  - Select *k* initial clusters randomly
  - Calculate the degree of membership of each data object to all clusters

$$\hat{w}_{li} = \begin{cases} 1, & \text{if } X_i = Z_l \\ 0, & \text{if } X_i = Z_h,\ h \neq l \\ \dfrac{1}{\sum_{h=1}^{k} \left[\dfrac{d(Z_l, X_i)}{d(Z_h, X_i)} \right]^{1/(\alpha-1)}}, & \text{if } X_i \neq Z_l \text{ and } X_i \neq Z_h,\ 1 \leq h \leq k \end{cases}$$

  - After all objects have been assigned with a degree of membership, update the new cluster centres based on the frequency of categories of attributes (modes of attributes)
  - The algorithm repeats until there is no change in the reassignment of objects or in the location of the centres

Fuzzy *k*-Modes Clustering (Cont'd)

Algorithm: **Fuzzy *k*-Modes clustering**

1. Select k initial clusters randomly $Z^{(1)} = \{Z_1^{(1)}, Z_2^{(1)}, \ldots, Z_k^{(1)}\}$.
2. Determine $W^{(1)}$ such that $F(W, Z^{(1)})$ is minimised.
3. Set $t = 1$.
4. Determine $Z^{(t+1)}$ satisfying for such that $F(W^{(t)}, Z^{(t+1)})$ is minimised.

   If $F(W^{(t)}, Z^{(t+1)}) = F(W^{(t)}, Z^{(t)})$ then stop; otherwise go to step 5.
5. Determine $W^{(t+1)}$ using the same equation used in step 2, such that $F(W^{(t+1)}, Z^{(t+1)})$ is minimised.

   If $F(W^{(t+1)}, Z^{(t+1)}) = F(W^{(t)}, Z^{(t+1)})$ then stop; otherwise set $t = t + 1$ and go to step 4.

Fuzzy *k*-Modes Clustering (Cont'd)

- Distance between an object and the centres is measured by simply matching the attributes of the dataset and storing them in a partition matrix

$$d(X_1, X_2) \equiv \sum_{j=1}^{m} \delta(x_{1j}, x_{2j}) \qquad \delta(x_{1j}, x_{2j}) = \begin{cases} 0, & x_{1j} = x_{2j} \\ 1, & x_{1j} \neq x_{2j} \end{cases}$$

X	1	2	3	4	5	6	7	8	9	10	11	12
C1	15	2	16	3	13	13	16	4	14	5	16	14
C2	6	16	11	13	5	11	6	12	7	14	10	12
C3	12	17	5	15	7	11	11	16	7	15	4	13
C4	11	11	10	11	13	2	14	11	11	12	10	3

Fuzzy *k*-Modes Clustering (Cont'd)

- Cluster centres are updated using a frequency-based method that computes the modal values of each attribute defining the objects in the dataset

X1	p	x	s	n	t	p	f	c	n	k	e	e
X2	e	x	s	v	t	a	f	c	b	k	e	c
X3	e	b	s	w	t	l	f	c	b	n	e	c
X4	p	x	y	w	t	p	f	c	n	n	e	e
X5	e	x	s	g	f	n	f	w	b	k	t	e

Z	e	x	s	w	t	p	f	c	b	k	e	e

Methodology

- Evolution
- Description
- Validation

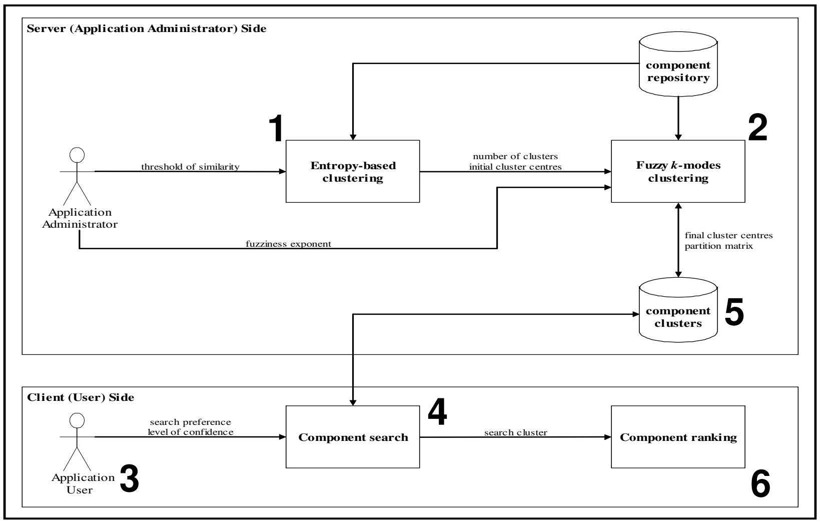
Evolution

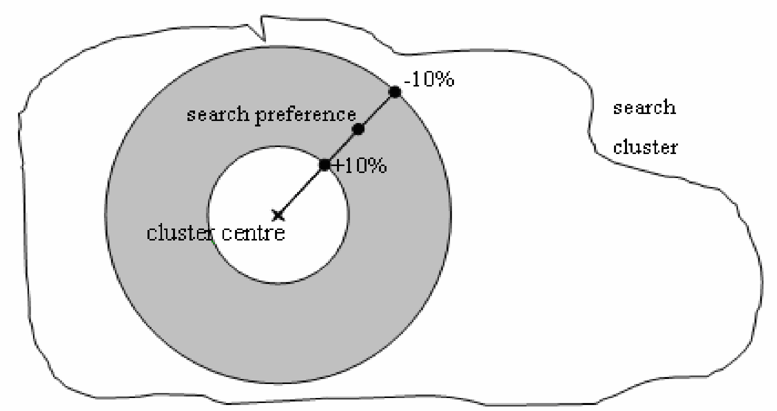
- Component clustering
  - Previous attempt with genetic algorithms
  - *k*-Means
  - *k*-Modes
- Limitations of *k*-modes with respect to software components clustering
  - ~~Inability to rank~~ use fuzzy *k*-modes instead
  - ~~Unknown value for *k*~~ use entropy-based clustering
  - ~~Sensitivity to initial cluster centres~~

Description

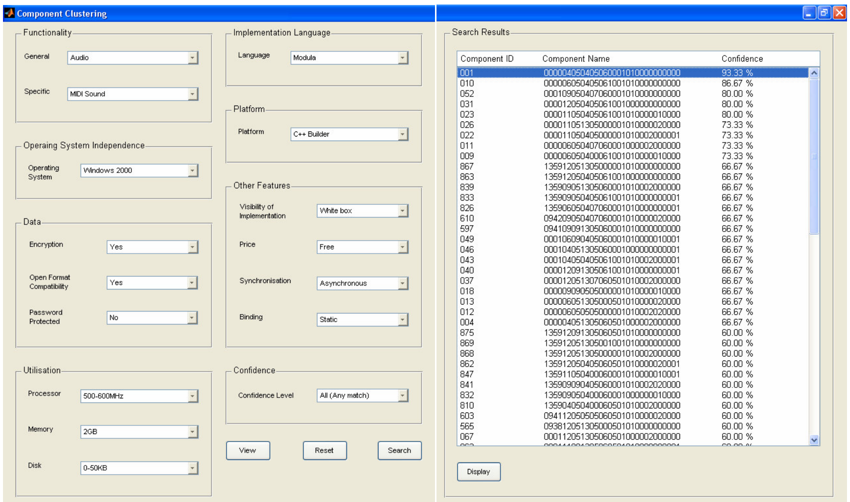
- Steps:
  1. Component clustering
     - Pre-processing (Entropy-based clustering)
     - Actual clustering (Fuzzy *k*-modes clustering)
  2. Isolate search cluster
     - Get user's search preference
     - Construct (closest) search cluster(s)
  3. Retrieve components
     - Assign search preference to search cluster(s)
     - Isolate range of membership degrees
     - Rank and display results to user

Methodology Scheme





Demonstration



Evaluation

- With a given a set of components, does the methodology adequately cluster the components and retrieve those most suitable based on a user's search preference?
- Design of Experiments
  - 1000 random components and 1 random search preference
  - Calculation of the similarity of all components with regards to the search preference
- The comparison was made according to the closest components in the dataset achieving similarity above:
  - 50% (near)
  - 75% (nearer)
  - 90% (nearest)

Evaluation (Cont'd)

- Performing searches using three different variations of the preference:
  - 15 attributes
  - 8 attributes
  - 4 attributes
- For entropy-based clustering: Threshold of similarity parameter $\beta = \{0.50, 0.55\}$
- For fuzzy *k*-modes clustering: Weighting exponent $\alpha = \{1.10, 1.20\}$

Evaluation (Cont'd)

- 15 attributes

Similarity to search preference > 50% Number of known near components **17**			
$\beta = 0.50$		$\beta = 0.55$	
$\alpha = 1.10$	$\alpha = 1.20$	$\alpha = 1.10$	$\alpha = 1.20$
13 (76%)	9 (53%)	11 (65%)	8 (47%)

Similarity to search preference > 75% Number of known nearer components **7**			
$\beta = 0.50$		$\beta = 0.55$	
$\alpha = 1.10$	$\alpha = 1.20$	$\alpha = 1.10$	$\alpha = 1.20$
7 (100%)	7 (100%)	7 (100%)	7 (100%)

Similarity to search preference > 90% Number of known nearest components **2**			
$\beta = 0.50$		$\beta = 0.55$	
$\alpha = 1.10$	$\alpha = 1.20$	$\alpha = 1.10$	$\alpha = 1.20$
2 (100%)	2 (100%)	2 (100%)	2 (100%)

Evaluation (Cont'd)

- 8 attributes

Similarity to search preference > 50% Number of known near components 45			
β = 0.50		β = 0.55	
α = 1.10	α =1.20	α = 1.10	α =1.20
12 (27%)	9 (20%)	10 (22%)	8 (18%)

Similarity to search preference > 75% Number of known nearer components 10			
β = 0.50		β = 0.55	
α = 1.10	α =1.20	α = 1.10	α =1.20
9 (90%)	8 (80%)	8 (80%)	8 (80%)

Similarity to search preference > 90% Number of known nearest components 2			
β = 0.50		β = 0.55	
α = 1.10	α =1.20	α = 1.10	α =1.20
2 (100%)	2 (100%)	2 (100%)	2 (100%)

Evaluation (Cont'd)

- 4 attributes

Similarity to search preference > 50% Number of known near components 87			
β = 0.50		β = 0.55	
α = 1.10	α =1.20	α = 1.10	α =1.20
15 (17%)	9 (10%)	11 (13%)	8 (9%)

Similarity to search preference > 75% Number of known nearer components 23			
β = 0.50		β = 0.55	
α = 1.10	α =1.20	α = 1.10	α =1.20
9 (39%)	8 (35%)	8 (35%)	7 (30%)

Similarity to search preference > 90% Number of known nearest components 7			
β = 0.50		β = 0.55	
α = 1.10	α =1.20	α = 1.10	α =1.20
7 (100%)	6 (86%)	6 (86%)	6 (86%)

Concluding Remarks

- Synopsis
- Pros and cons
- Future work

Synopsis

- Component-based software engineering builds large and complex software systems from small, autonomous and reusable pieces
- Process can be time-consuming due to discovery and evaluation of components
- Introduction of component repositories to store and organise software components → the need for techniques to search and retrieve software components from repositories
- Methodology reduces the time to locate components for reuse by using a hybrid clustering scheme using an entropy-based fuzzy *k*-modes clustering algorithm

Pros and Cons

- Pros:
  - Accurate
  - Efficient
  - Effective
  - Flexible
  - Expandable
  - Simple
  - Low-demanding in terms of number of inputs
  - High quality
- Cons:
  - The two parameters it relies on can significantly change search results
    - Subjectivity of the clustering process
  - Too many clusters created by entropy-based clustering

Future Work

- Attempt to apply the methodology to a real component repository and implemented with client-server model
- Modification/refinement of component features
- Try to find a way to automatically calculate parameters:
  - threshold of similarity
  - Weighting (fuzziness) exponent

Overall Summary - Discussion

Thank you for your attention.
Questions?

Department of Computer Science
University of Cyprus